Implementing field extensions of the form $\mathbb{Q}[\sqrt{b}]^*$

René Thiemann

April 20, 2020

Abstract

We apply data refinement to implement the real numbers, where we support all numbers in the field extension $\mathbb{Q}[\sqrt{b}]$, i.e., all numbers of the form $p + q\sqrt{b}$ for rational numbers $p$ and $q$ and some fixed natural number $b$. To this end, we also developed algorithms to precisely compute roots of a rational number, and to perform a factorization of natural numbers which eliminates duplicate prime factors.

Our results have been used to certify termination proofs which involve polynomial interpretations over the reals.

Contents

1 Introduction 1
2 Auxiliary lemmas which might be moved into the Isabelle distribution. 2
3 Prime products 2
4 A representation of real numbers via triples 9
5 A unique representation of real numbers via triples 19

1 Introduction

It has been shown that polynomial interpretations over the reals are strictly more powerful for termination proving than polynomial interpretations over the rationals. To this end, also automated termination prover started to generate such interpretations. [3, 4, 5, 7, 8]. However, for all current implementations, only reals of the form $p + q \cdot \sqrt{b}$ are generated where $b$ is some fixed natural number and $p$ and $q$ may be arbitrary rationals, i.e., we get numbers within $\mathbb{Q}[\sqrt{b}]$.

*This research is supported by FWF (Austrian Science Fund) project P22767-N13.
To support these termination proofs in our certifier CéTa [6], we therefore required executable functions on $\mathbb{Q}[\sqrt{b}]$, which can then be used as an implementation type for the reals. Here, we used ideas from [1, 2] to provide a sufficiently powerful partial implementations via data refinement.

2 Auxiliary lemmas which might be moved into the Isabelle distribution.

theory Real-Impl-Auxiliary
import HOL-Computational-Algebra.Primes
begin

lemma multiplicity-prime:
assumes $p$: prime $(i :: nat)$ and $ji$: $j \neq i$
shows $\text{multiplicity } j i = 0$
using assms
by (metis dvd-refl prime-nat-iff multiplicity-eq-zero-iff
multiplicity-unit-left multiplicity-zero)

end

3 Prime products

theory Prime-Product
import Real-Impl-Auxiliary
Sqrt-Babylonian
begin

Prime products are natural numbers where no prime factor occurs more than once.

definition prime-product
where prime-product $(n :: nat) = (\forall p. \text{prime } p \rightarrow \text{multiplicity } p n \leq 1)$

The main property is that whenever $b_1$ and $b_2$ are different prime products, then $p_1 + q_1\sqrt{b_1} = p_2 + q_2\sqrt{b_2}$ implies $(p_1, q_1, b_1) = (p_2, q_2, b_2)$ for all rational numbers $p_1, q_1, p_2, q_2$. This is the key property to uniquely represent numbers in $\mathbb{Q}[\sqrt{b}]$ by triples. In the following we develop an algorithm to decompose any natural number $n$ into $n = s^2 \cdot p$ for some $s$ and prime product $p$.

function prime-product-factor-main :: nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\times$ nat
where
prime-product-factor-main factor-sq factor-pr limit n i =
(if $i \leq \text{limit}$ $\land$ $i \geq 2$ then
  (if $i \text{ dvd } n$

2
then (let n' = n div i in
  (if i dvd n' then
    let n'' = n' div i in
    prime-product-factor-main (factor-sq * i) factor-pr (nat (root-nat-floor 3 n'')) n'' i
  else
    (case sqrt-nat n' of
      Cons sn - ⇒ (factor-sq * sn, factor-pr * i)
      | [] ⇒ prime-product-factor-main-factor-sq (factor-pr * i) (nat (root-nat-floor 3 n')) n' (Suc i)
    )
  )
) else
prime-product-factor-main-factor-sq factor-pr limit n (Suc i)
else
(factor-sq, factor-pr * n)) by pat-completeness auto

termination
proof –
let ?m1 = λ (factor-sq :: nat, factor-pr :: nat, limit :: nat, n :: nat, i :: nat). n
let ?m2 = λ (factor-sq, factor-pr, limit, n, i). (Suc limit - i)
{ fix i
  have 2 ≤ i ⇒ Suc 0 < i * i using one-less-mult[of i i] by auto
} note * = this
show ?thesis
  using wf-measures [of [?m1, ?m2]]
  by rule (auto simp add: * elim!: dvdE split: if-splits)
qed

lemma prime-product-factor-main: assumes ¬ (∃ s. s * s = n)
  and limit = nat (root-nat-floor 3 n)
  and m = factor-sq * factor-sq * factor-pr * n
  and prime-product-factor-main-factor-sq factor-pr-factor-pr limit n i = (sq, p)
  and i ≥ 2
  and ∏ j. j ≥ 2 ⇒ j < i ⇒ ¬ j dvd n
  and ∏ j. prime j ⇒ j < i ⇒ multiplicity j factor-pr ≤ 1
  and ∏ j. prime j ⇒ j ≥ i ⇒ multiplicity j factor-pr = 0
  and factor-pr > 0
  shows m = sq * sq * p ∧ prime-product p
using assms
proof (induct factor-sq factor-pr limit n i rule: prime-product-factor-main.induct)
case (1 factor-sq factor-pr limit n i)
note IH = 1(1–3)
note prems = 1(4–)
  note simp = prems(4)[unfolded prime-product-factor-main.simps[of factor-sq factor-pr limit n i]]
show ?case

3
proof (cases i ≤ limit)
  case True note i = this
  with prems(5) have cond: i ≤ limit ∧ i ≥ 2 and \*: (i ≤ limit ∧ i ≥ 2) = True by blast+
    note IH = IH[OF cond]
    note simp = simp[unfolded * if-True]
    show \?thesis
proof (cases i dvd n)
  case False
    hence \*: (i dvd n) = False by simp
    note IH = IH(3)[OF False prems(1-3) simp]
    show \?thesis
proof (rule IH)
  fix j
    assume 2: 2 ≤ j and j: j < Suc i
    from prems(6)[OF 2] j False
    show ¬ j dvd n by (cases j = i, auto)
next
  fix j :: nat
    assume j: j < Suc i prime j
    with prems(7-8) [of j]
    show multiplicity j factor-pr ≤ 1 by (cases j = i, auto)
qed (insert prems(8-9) cond, auto)
next
  case True note mod = this
    hence (i dvd n) = True by simp
    note simp = simp[unfolded this if-True Let-def]
    note IH = IH(1,2)[OF True refl]
    show \?thesis
proof (cases i dvd n div i)
  case True
    hence \*: (i dvd n div i) = True by auto
    define n' where n' = n div i div i
    from mod True have n: n = n' * i * i by (auto simp: n'-def dvd-eq-mod-eq-0)
    note simp = simp[unfolded * if-True split]
    note IH = IH(1)[OF True refl - refl - simp prems(5) - prems(7-9)]
    show \?thesis
proof (rule IH)
  show m = factor-sq * i * (factor-sq * i) * factor-pr * (n div i div i)
    unfolding prems(3) n'-def[symmetric]
    unfolding n by (auto simp: field-simps)
next
  fix j
    assume 2: 2 ≤ j j < i
    from prems(6)[OF this] have ¬ j dvd n by auto
    thus ¬ j dvd n div i div i
      by (metis dvd-mult n n'-def mult.commute)
next
show \(\neg (\exists s. s \ast s = n \div i \div i)\)

proof
  assume \(\exists s. s \ast s = n \div i \div i\)
  then obtain s where \(s \ast s = n \div i \div i\) by auto
  hence \((s \ast i) \ast (s \ast i) = n\) unfolding \(n\) by auto
  with prems(1) show \(False\) by blast
qed
qed

next
case \(False\)
define \(n' = n \div i\)
from mod True have \(n = n' \ast i\) by (auto simp: \(n'\)-def dvd-eq-mod-eq-0)
have prime: prime \(i\)
  unfolding prime-nat-iff
proof (intro conjI allI impI)
fix \(m\)
  assume \(m: m \ dvd n\)
  hence \(m \ dvd n\) unfolding \(n\) by auto
  with prems(6)[of \(m\)] have choice: \(m \leq 1 \lor m \geq i\) by arith
  from \(m\) prems(5) have \(m > 0\)
    by (metis dvd-0-left-iff le0 le-antisym neq0-conv zero-neq-numeral)
  with choice have choice: \(m = 1 \lor m \geq i\) by arith
  from \(m\) prems(5) have \(m \leq i\)
    by (metis False dvd-by-0 dvd-refl dvd-imp-le gr0I)
  with choice
  show \(m = 1 \lor m = i\) by auto
qed (insert prems(5), auto)
from \(False\) have \((i \ dvd n \ div i) = False\) by auto
note simp = simp[unfolded this if-False]
note IH = IH(2)[OF \(False\) - - refl]
from prime have \(i > 0\) by (simp add: prime-gt-0-nat)

show \(?thesis\)
proof (cases sqrt-nat \((n \ div i)\))
case (Cons \(s\))
  note simp = simp[unfolded Cons list.simps]
hence sq: \(sq = \text{factor-sq} \ast s\) and \(p: p = \text{factor-pr} \ast i\) by auto
from arg-cong[OF Cons, of set] have \(s: s \ast s = n \ div i\) by auto
have pp: prime-product \((\text{factor-pr} \ast i)\)
  unfolding prime-product-def
proof safe
fix \(m::\text{nat}\) assume \(m: \text{prime } m\)
  consider \(i < m \mid i > m \mid i = m\) by force
  thus multiplicity \(m\) \((\text{factor-pr} \ast i)\) \(\leq 1\)
    by cases (insert prems(7)[of \(m\)]) prems(8)[of \(m\)] prems(9) \(i > 0\): prime \(m\),
      simp-all add: multiplicity-prime prime-elem-multiplicity-mult-distrib)
qed
show \(?thesis\) unfolding sq p prems(3) \(n\) unfolding \(n'\)-def \(s\)[symmetric]
using pp by auto

next
case Nil
  note simp = simp[unfolded Nil, list.simps]
from arg-cong[OF Nil, of set] have \( n \in \{x. x * x = n \div i\} \) by simp
note IH = IH[OF Nil this - simp]
show \( ?thesis \)
proof (rule IH)
  show \( m = \text{factor-sq} \times \text{factor-sq} \times (\text{factor-pr} \times i) \times (n \div i) \)
    unfolding prems(3) n by auto
next
  fix j
  assume : \( 2 \leq j < \text{Suc } i \)
  show \( j \not\mid n \div i \)
  proof
    assume j: \( j \not\mid n \div i \)
    with False have \( j \neq i \) by auto
    with * have \( 2 \leq i < j \) by auto
    from prems(6)[OF this] j
    show False unfolding n
    by (metis dvd-mult n n')
  qed
next
  fix j :: nat
  assume Suc i \leq j and j-prime: prime j
  hence ij: \( i \leq j \) and j: \( j \neq i \) by auto
  have 0: multiplicity j i = 0 using prime j by (rule multiplicity-prime)
  show multiplicity j (factor-pr \times i) = 0
    unfolding prems(8)[OF j-prime ij] 0
    using prime j-prime j \times 0 < factor-pr \times multiplicity j factor-pr = 0
    by (subst prime-elem-multiplicity-mult-distrib) (auto simp: multiplicity-prime)
next
  fix j
  assume j < Suc i and j-prime: prime j
  hence j < i \lor j = i by auto
  thus multiplicity j (factor-pr \times i) \leq 1
  proof
    assume j = i
    with prems(8)[of i] prime j-prime \times 0 < factor-pr \show ?thesis
    by (subst prime-elem-multiplicity-mult-distrib) auto
next
  assume ji: j < i
  hence j \neq i by auto
  from prems(7)[OF j-prime ji] multiplicity-prime[OF prime this]
  prime j-prime \times 0 < factor-pr
  show ?thesis by (subst prime-elem-multiplicity-mult-distrib) auto
qed
qed (insert prems(5,9), auto)
qed
qed
next

case False

hence \((i \leq \text{limit} \land i \geq 2) = False\) by auto

note simp = simp[unfolded this if-False]

hence sq: sq = factor-sq and p: p = factor-pr * n by auto

show ?thesis

proof

show \(m = sq * sq * p\) unfolding sq p prems(3) by simp

show prime-product p unfolding prime-product-def

proof safe

fix \(m :: \text{nat}\) assume \(m: \text{prime} m\)

from prems(1) have \(n1: n > 1\) by (cases n, auto, case-tac nat, auto)

hence \(n0: n > 0\) by auto

have \(i > \text{limit}\) using False by auto

from this[unfolded prems(2)] have less: \(\text{int} i \geq \text{root-nat-floor} 3 \ n + 1\) by auto

have \(\text{int} n < (\text{root-nat-floor} 3 \ n + 1) ^ 3\) by (rule root-nat-floor-upper, auto)

also have \(\ldots \leq \text{int} i \ ^ 3\) by (rule power-mono[of less, of 3], auto)

finally have \(\ldots \leq \text{int} i  ^ 3\) by (rule power-mono[of less, of 3], auto)

{ fix \(m\)

assume \(m: \text{prime} m\) multiplicity \(m \ n > 0\)

hence \(mp: m \in \text{prime-factors} n\)

by (auto simp: prime-factors-multiplicity)

hence \(md: m \ dvd \ n\)

by auto

then obtain \(k\) where \(n: n = m \ast k\) ..

from \(mp\) have \(pm: \text{prime} m\) by auto

hence \(m2: m \geq 2\) and \(m0: m > 0\) by (auto simp: prime-nat-iff)

from prems(6)[OF m2] have \(mi: m \geq i\) by force

{ assume multiplicity \(m \ n \neq 1\)

with \(m\) have \(\exists k. \text{multiplicity} \ m \ n = 2 + k\) by presburger

then obtain \(j\) where \(\text{mult: multiplicity} \ m \ n = 2 + j\) ..

from \(n0\) \(n\) have \(k: k > 0\) by auto

from \(\text{mult} \ m0\) \(k \ m\) have \(\text{multiplicity} \ m \ k > 0\)

by (auto simp: prime-elem-multiplicity-mult-distrib)

with \(m\) have \(mp: m \in \text{prime-factors} k\)

by (auto simp: prime-factors-multiplicity)

hence \(md: m \ dvd \ k\) by (auto simp: k)

then obtain \(l\) where \(\text{kml: k = m \ast l}\)

note \(n = n[\text{unfolded kml}]\)

from \(n\) have \(\text{l dvd} \ n\) by auto

with prems(6)[of \(l\)] have \(l \leq 1 \lor l \geq i\) by arith

with \(n \ n0\) have \(\text{l}: l = 1 \lor l \geq i\) by auto

7
from n prems(1) have \( l \neq 1 \) by auto
with \( l \) have \( l \geq i \) by auto
from mult-le-mono[OF mult-le-mono[OF mi mi] \( l \) \( ] \) have \( n \geq i^3 \) unfolding \( n \) by (auto simp: power3-eq-cube)
with \( n-i^3 \) have False by auto
}

with \( mi \ m \) have multiplicity \( m \ n = 1 \land m \geq i \) by auto
}

note \( n = \) this
have multiplicity \( m \ p = \) multiplicity \( m \) factor-pr + multiplicity \( m \) \( n \)
unfolding \( p \) using prems(1,9) \( \{ n > 0 \} \)
by (auto simp: prime-elem-multiplicity-mult-distrib)
also have \( \ldots \leq 1 \)
proof (cases \( m < i \))
case True
from prems(7)[of \( m \)] \( n[of \ m] \) True \( m \) show ?thesis by force
next
case False
hence \( m \geq i \) by auto
from prems(8)[OF \( m(1) \) \( \{ \) \( \) this] \( n[of \ m] \) \( m \) show ?thesis by force
qed
finally show multiplicity \( m \) \( p \leq 1 \).
qed

definition prime-product-factor :: \( \text{n \Rightarrow \text{n \times n}} \) where
prime-product-factor \( n = (\text{case sqrt-nat \( n \) of}
(\text{Cons \( s \) -}) \Rightarrow (s,1)
| [] \Rightarrow \text{prime-product-factor-main \( 1 \ 1 \) (nat (root-nat-floor 3 \( n \)) \( n \) 2)}

lemma prime-product-one[simp, intro]: prime-product 1
unfolding prime-product-def multiplicity-one-nat by auto

lemma prime-product-factor: assumes \( \text{pf} \): prime-product-factor \( n = (\text{sq,p}) \)
shows \( n = \text{sq * sq * p} \land \text{prime-product \( p \)\)}
proof (cases \( \text{sqrt-nat \( n \)} \))
case (\text{Cons \( s \) })
from \( \text{pf}\) unfolded prime-product-factor-def Cons arg-cong[OF Cons, of set]
prime-product-one
show ?thesis by auto
next
case Nil
from arg-cong[OF Nil, of set] have \( \text{nsq} \): \( \lnot (\exists \ s. \ s * s = n) \) by auto
show ?thesis
by (rule prime-product-factor-main[OF \( \text{nsq refl, of - 1 1 2} \)], unfold multiplicity-one,
insert \( \text{pf} \) unfolded prime-product-factor-def Nil, auto)

8
We represent real numbers of the form \( p + q \cdot \sqrt{b} \) for \( p, q \in \mathbb{Q}, n \in \mathbb{N} \) by triples \((p, q, b)\). However, we require the invariant that \( \sqrt{b} \) is irrational. Most binary operations are implemented via partial functions where the common restriction is that the numbers \( b \) in both triples have to be identical. So, we support addition of \( \sqrt{2} + \sqrt{2} \), but not \( \sqrt{2} + \sqrt{3} \).

The set of natural numbers whose square root is irrational

\[
\text{definition sqrt-irrat} = \{ q :: \text{nat}. \neg (\exists p. p \cdot p = \text{rat-of-nat} q) \}
\]

\[
\text{lemma sqrt-irrat: assumes choice: } q = 0 \lor b \in \text{sqrt-irrat}
\]

\[
\text{ and eq: real-of-rat } p + \text{real-of-rat } q \cdot \sqrt{\text{of-nat } b} = 0
\]

\[
\text{shows } q = 0
\]

\[
\text{using choice}
\]

\[
\text{proof (cases } q = 0 \text{)}
\]

\[
\text{case False}
\]

\[
\text{with choice have sqrt-irrat: } b \in \text{sqrt-irrat by blast}
\]

\[
\text{from eq have real-of-rat } q \cdot \sqrt{\text{of-nat } b} = \text{real-of-rat } (\neg p)
\]

\[
\text{by (auto simp: of-rat-minus)}
\]

\[
\text{then obtain } p \text{ where real-of-rat } q \cdot \sqrt{\text{of-nat } b} = \text{real-of-rat } p \text{ by blast}
\]

\[
\text{from arg-cong[OF this, of } \lambda x. x \cdot x] \text{ have real-of-rat } (q \cdot q) \cdot (\sqrt{\text{of-nat } b})
\]

\[
\text{* sqrt (of-nat } b) = \text{real-of-rat } (p \cdot p) \text{ by (auto simp: field-simps of-rat-mult)}
\]

\[
\text{also have sqrt (of-nat } b) \cdot \sqrt{\text{of-nat } b} = \text{of-nat } b \text{ by simp}
\]

\[
\text{finally have real-of-rat } (q \cdot q \cdot \text{rat-of-nat } b) = \text{real-of-rat } (p \cdot p) \text{ by (auto simp: of-rat-mult)}
\]

\[
\text{hence } q \cdot q \cdot (\text{rat-of-nat } b) = p \cdot p \text{ by auto}
\]

\[
\text{from arg-cong[OF this, of } \lambda x. x / (q \cdot q)] \text{ have } (p / q) \cdot (p / q) = \text{rat-of-nat } b \text{ using False by (auto simp: field-simps)}
\]

\[
\text{with sqrt-irrat show } \text{thesis unfolding sqrt-irrat-def by blast}
\]

\[
\text{qed}
\]

To represent numbers of the form \( p + q \cdot \sqrt{b} \), use mini algebraic numbers, i.e., triples \((p, q, b)\) with irrational \( \sqrt{b} \).

\[
\text{typedef mini-alg} =
\]

\[
\{(p, q, b) \mid p :: \text{rat} \quad q :: \text{rat} \quad b :: \text{nat}.
\]

\[
q = 0 \lor b \in \text{sqrt-irrat}
\]

\[
\text{by auto}
\]
setup-lifting type-definition-mini-alg

lift-definition real-of :: mini-alg ⇒ real is
  λ (p,q,b). of-rat p + of-rat q * sqrt (of-nat b) .

lift-definition ma-of-rat :: rat ⇒ mini-alg is λ x. (x,0,0) by auto

lift-definition ma-rat :: mini-alg ⇒ rat is fst .
lift-definition ma-base :: mini-alg ⇒ nat is snd o snd .
lift-definition ma-coeff :: mini-alg ⇒ rat is fst o snd .

lift-definition ma-uminus :: mini-alg ⇒ mini-alg is
  λ (p1,q1,b1). (− p1, − q1, b1) by auto

lift-definition ma-compatible :: mini-alg ⇒ mini-alg ⇒ bool is
  λ (p1,q1,b1) (p2,q2,b2). q1 = 0 ∨ q2 = 0 ∨ b1 = b2 .

definition ma-normalize :: rat × rat × nat ⇒ rat × rat × nat where
  ma-normalize x ≡ case x of (a,b,c) ⇒ if b = 0 then (a,0,0) else (a,b,c)

lemma ma-normalize-case[simp]: (case ma-normalize r of (a,b,c) ⇒ real-of-rat a + real-of-rat b * sqrt (of-nat c))
  = (case r of (a,b,c) ⇒ real-of-rat a + real-of-rat b * sqrt (of-nat c))
by (cases r, auto simp: ma-normalize-def)

lift-definition ma-plus :: mini-alg ⇒ mini-alg ⇒ mini-alg is
  λ (p1,q1,b1) (p2,q2,b2). if q1 = 0 then
  (p1 + p2, q2, b2) else ma-normalize (p1 + p2, q1 + q2, b1) by (auto simp: ma-normalize-def)

lift-definition ma-times :: mini-alg ⇒ mini-alg ⇒ mini-alg is
  λ (p1,q1,b1) (p2,q2,b2). if q1 = 0 then
  ma-normalize (p1*p2, p1*q2, b2) else
  ma-normalize (p1*p2 + of-nat b2*q1+q2, p1*q2 + q1*p2, b1) by (auto simp: ma-normalize-def)

lift-definition ma-inverse :: mini-alg ⇒ mini-alg is
  λ (p,q,b). let d = inverse (p * p − of-nat b * q + q) in
  ma-normalize (p * d, − q * d, b) by (auto simp: Let-def ma-normalize-def)

lift-definition ma-floor :: mini-alg ⇒ int is
  λ (p,q,b). case (quotient-of p,quotient-of q) of ((z1,n1),(z2,n2)) ⇒
  let z2n1 = z2 * n1; z1n2 = z1 * n2; n12 = n1 + n2; prod = z2n1 * z2n1 * int b in
  (z1n2 + (if z2n1 ≥ 0 then sqrt-int-floor-pos prod else − sqrt-int-ceiling-pos prod)) div n12 .

lift-definition ma-sqrt :: mini-alg ⇒ mini-alg is
  λ (p,q,b). let (a,b) = quotient-of p; aa = abs (a * b) in
case sqrt-int aa of [] ⇒ (0, inverse (of-int b), nat aa) | (Cons s -) ⇒ (of-int s / of-int b, 0, 0)
proof (unfold Let-def)
  fix prod :: rat × rat × nat
obtain a b where p: quotient-of p = (a, b) by force
show ∃ p q b. (case prod of
  (p, q, b) ⇒
  case quotient-of p of
  (a, b) ⇒
  (case sqrt-int |a * b| of [] ⇒ (0, inverse (of-int b), nat |a * b|)
  | s # x ⇒ (of-int s / of-int b, 0, 0))) =
  (p, q, b) ∧
  (q = 0 ∨ b ∈ sqrt-irrat)
proof (cases sqrt-int (abs (a * b)))
case Nil
from sqrt-int[of abs (a * b)] Nil have irrat: ¬ (∃ y. y * y = |a * b|) by auto
have nat |a * b| ∈ sqrt-irrat
proof (rule ccontr)
  assume nat |a * b| / ∈ sqrt-irrat
  then obtain x :: rat
  let bqq = of-nat b * q * q
  pp = p * p
  0 ≤ p ∧ bqq ≤ pp ∨ 0 ≤ q ∧ pp ≤ bqq.
qed
  thus thesis using Nil by (auto simp: prod p)
qed (auto simp: prod p Cons)
qed

lift-definition ma-equal :: mini-alg ⇒ mini-alg ⇒ bool is
  λ(p1,q1,b1) (p2,q2,b2).
  p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2).

lift-definition ma-ge-0 :: mini-alg ⇒ bool is
  λ(p,q,b), let bqq = of-nat b * q * q; pp = p * p
  0 ≤ p ∧ bqq ≤ pp ∨ 0 ≤ q ∧ pp ≤ bqq.

lift-definition ma-is-rat :: mini-alg ⇒ bool is
  λ(p,q,b). q = 0.

definition ge-0 :: real ⇒ bool where [code del]: ge-0 x = (x ≥ 0)

lemma ma-ge-0: ge-0 (real-of x) = ma-ge-0 x
proof (transfer, unfold Let-def, clarsimp)
  fix p' q' :: rat and b' :: nat
  assume b': q' = 0 ∨ b' ∈ sqrt-irrat
  define b where b = real-of-nat b'
  define p where p = real-of-rat p'
  define q where q = real-of-rat q'
from $b'$ have $b: 0 \leq b \leq q = 0 \lor b' \in \sqrt{\text{irrat}}$ unfolding $b$-def
$q$-def by auto

define $qb$ where $qb = q * \sqrt{b}$

from $b$ have $\sqrt{b}: \sqrt{b} \geq 0$ by auto

from $b(2)$ have $\text{disj}: q = 0 \lor b \neq 0$ unfolding $\sqrt{\text{irrat}}$-def $b$-def by auto

have $b: \text{def}: b = \text{real-of-rat} (\text{of-nat} b')$ unfolding $b$-def by auto

have $(0 \leq p + q * \sqrt{b}) = (0 \leq p + qb)$ unfolding $qb$-def by simp

also have $\ldots \iff (0 \leq p \land \text{abs} \; q \leq \text{abs} \; p \lor 0 \leq q \land \text{abs} \; p \leq \text{abs} \; qb)$ by arith

also have $\ldots \iff (0 \leq p \land q \leq p \lor 0 \leq qb \land p \leq qb \land qb)$ unfolding $\text{abs-less-eq-square}$. ..

also have $qb \leq q = b * q$ unfolding $qb$-def using $\text{sqrt diseq}$

by $(\text{metis le-cases mult-eq-0-iff mult-nonneg-nonneg \text{mult-nonpos-nonneg order-class. order.antisym}$
$qb$-def real-sqrt-eq-zero-cancel-iff $)$

also have $(0 \leq p \land b * q \leq p \land 0 \leq q \land b * p \leq b * q \land q)$

$\iff (0 \leq p' \land 0 \leq q \land q' \leq p' \land 0 \leq q' \land p' \leq of-nat b' * q' \land q')$ unfolding $qb$-def $q$-def

by $(\text{unfold bdef p-def q-def of-rat-mult[symmetric] of-rat-less-eq, simp})$

finally

show $\text{ge-0} (\text{real-of-rat} p' + \text{real-of-rat} q' * \sqrt{\text{of-nat} b'}) =$

$(0 \leq p' \land 0 \leq q' \land q' \leq p' \land 0 \leq q' \land p' \leq of-nat b' * q' * q')$ unfolding $\text{ge-0-def}$ $p$-def $b$-def $q$-def

by $(\text{auto simp: of-rat-add of-rat-mult})$

qed

lemma $\text{ma-0: 0 = real-of (ma-of-rat 0)}$ by (transfer, auto)

lemma $\text{ma-1: 1 = real-of (ma-of-rat 1)}$ by (transfer, auto)

lemma $\text{ma-uminus:}$

$-(\text{real-of x}) = \text{real-of (ma-uminus x)}$

by $(\text{transfer, auto simp: of-rat-minus})$

lemma $\text{ma-inverse: inverse (real-of r) = real-of (ma-inverse r)}$

proof $(\text{transfer, unfold Let-def, clarsimp})$

fix $p' \; q' :: \text{rat and b'} :: \text{nat}$

assume $b': q' = 0 \lor b' \in \sqrt{\text{irrat}}$

define $b$ where $b = \text{real-of-nat} b'$

define $p$ where $p = \text{real-of-rat} p'$

define $q$ where $q = \text{real-of-rat} q'$

from $b'$ have $b: b \geq 0 \land q = 0 \lor b' \in \sqrt{\text{irrat}}$ unfolding $b$-def $q$-def by auto

have $\text{inverse} (p + q * \sqrt{b}) = (p - q * \sqrt{b}) * \text{inverse} (p * p - b * (q * q))$

proof $(\text{cases q = 0})$

case $\text{True}$ thus $\text{thesis by} \ (\text{cases p = 0, auto simp: field-simps})$

next

case $\text{False}$

from $\sqrt{\text{irrat}}(\text{OF} b', \text{of} p')$ $b$-def $p$-def $q$-def False have $\text{nnull: p + q * sqrt b}$. }
≠ 0 by auto

have ?thesis ⟷ (p + q * sqrt b) * inverse (p + q * sqrt b) =
(p + q * sqrt b) * ((p − q * sqrt b) * inverse (p * p − b * (q * q)))

unfolding mult-left-cancel[OF nnull] by auto

also have (p + q * sqrt b) * inverse (p + q * sqrt b) = 1 using nnull by auto
also have (p + q * sqrt b) * ((p − q * sqrt b) * inverse (p * p − b * (q * q)))
= (p * p − b * (q * q)) * inverse (p * p − b * (q * q))

using b by (auto simp: field-sims)

also have ... = 1

proof (rule right-inverse, rule)
assume eq: p * p − b * (q * q) = 0
have real-of-rat (p′ * p′ / (q′ * q′)) = p * p / (q * q)
  unfolding p-def b-def q-def by (auto simp: of-rat-mult of-rat-divide)
also have ... = b using eq False by (auto simp: field-sims)
also have ... = real-of-rat (of-nat b′) unfolding b-def by auto
finally have (p′ / q′) * (p′ / q′) = of-nat b′

unfolding of-rat-eq-iff by simp
with b False show False unfolding sqrt-irrat-def by blast
qed

finally
show ?thesis by simp
qed

thus inverse (real-of-rat p′ + real-of-rat q′ * sqrt (of-nat b′)) =
real-of-rat (p′ * inverse (p′ − of-nat b′ * q′ * q′)) +
real-of-rat (− (q′ * inverse (p′ − of-nat b′ * q′ * q′)) * sqrt (of-nat b′))

by (simp add: divide-sims of-rat-mult of-rat-divide of-rat-diff of-rat-minus b-def
p-def q-def
  split: if-splits)
qed

lemma ma-sqrt-main: ma-rat r ≥ 0 ⇒ ma-coeff r = 0 ⇒ sqrt (real-of r) =
real-of (ma-sqrt r)
proof (transfer, unfold Let-def, clarsimp)
fix p :: rat
assume p: 0 ≤ p
hence abs: abs p = p by auto

obtain a b where ab: quotient-of p = (a,b) by force
hence pab: p = of-int a / of-int b by (rule quotient-of-denom-div)

from ab have b: b > 0 by (rule quotient-of-denom-pos)
with p pab have abpos: a * b ≥ 0
  by (metis of-int-0-le-iff of-int-le-0-iff zero-le-divide-iff zero-le-mult-iff)

have rab: of-nat (nat (a * b)) = real-of-int a * real-of-int b using abpos
  by simp

let ?lhs = sqrt (of-int a / of-int b)

let ?rhs = (case case quotient-of p of
  (a, b) ⇒ (case sqrt-int |a |b| of [] ⇒ (0, inverse (of-int b), nat |a * b|)
     | s ≠ x ⇒ (of-int s / of-int b, 0, 0)) of
  (p, q, b) ⇒ of-rat p + of-rat q * sqrt (of-nat b))
have sqrt (real-of-rat p) = ?lhs unfolding pab
  by (metis of-rat-divide of-rat-of-int-eq)
also have ... = ?rhs
proof (cases sqrt-int | a * b)
  case Nil
  define sb where sb = sqrt (of-int b)
  define sa where sa = sqrt (of-int a)
  from b sb-def have sb: sb > 0 real-of-int b > 0 by auto
  have sbb: sb * sb = real-of-int b unfolding sb-def
    by (rule sqrt-sqrt, insert b, auto)
  from Nil have ?thesis = (sa / sb = inverse (of-int b) * (sa * sb)) unfolding ab def using ab
  by (metis b divide-real-def eq-divide-imp inverse-divide inverse-inverse-eq inverse-mult-distrib less-int-code(1) of-int-eq-0-iff real-sqrt-zero-cancel-iff sb-def sbb times-divide-eq-right)
  also have ... = True using sb
next
  case (Cons s x)
  from b have b: real-of-int b > 0 by auto
  from Cons sqrt-int[of abs (a * b)] have s * s = abs (a * b) by auto
  with sqrt-int-pos[OF Cons] have sqrt (real-of-int (abs (a * b))) = of-int s
  by (metis abs-of-nonneg of-int-mult of-int-abs real-sqrt-abs2)
  with abpos have of-int s = sqrt (real-of-int (a * b)) by auto
  thus ?thesis unfolding ab split using Cons b
    by (auto simp: of-rat-divide field-simps real-sqrt-divide real-sqrt-mult)
qed

lemma ma-sqrt: sqrt (real-of r) = (if ma-coeff r = 0 then
  (if ma-rat r ≥ 0 then real-of (ma-sqrt (ma-rat r)) else - real-of (ma-sqrt (ma-uminus r)))))
  else Code.abort (STR "cannot represent sqrt of irrational number") (λ _. sqrt (real-of r)))
proof (cases ma-coeff r = 0)
  case True note 0 = this
  hence 00: ma-coeff (ma-uminus r) = 0 by (transfer, auto)
  show ?thesis
proof (cases ma-rat r ≥ 0)
  case True
  from ma-sqrt-main[OF this 0] 0 True show ?thesis by auto
next
  case False
  hence ma-rat (ma-uminus r) ≥ 0 by (transfer, auto)
  from ma-sqrt-main[OF this 00, folded ma-uminus, symmetric] False 0
  show ?thesis by (auto simp: real-sqrt-minus)
qed

lemma ma-plus:
\[(\text{real-of } r1 + \text{real-of } r2) = (\text{if ma-compatible } r1 \ r2 \text{ then real-of (ma-plus } r1 \ r2) \text{ else Code.abort (STR "different base") } (\lambda -. \text{real-of } r1 + \text{real-of } r2))\]
by transfer (auto split: prod.split simp: field-simps of-rat-add)

lemma ma-times:
\[(\text{real-of } r1 \ast \text{real-of } r2) = (\text{if ma-compatible } r1 \ r2 \text{ then real-of (ma-times } r1 \ r2) \text{ else Code.abort (STR "different base") } (\lambda -. \text{real-of } r1 \ast \text{real-of } r2))\]
by transfer (auto split: prod.split simp: field-simps of-rat-mult of-rat-add)

lemma ma-equal:
HOL.equal (\text{real-of } r1) (\text{real-of } r2) = (\text{if ma-compatible } r1 \ r2 \text{ then ma-equal } r1 \ r2 \text{ else Code.abort (STR "different base") } (\lambda -. \text{HOL.equal } (\text{real-of } r1) (\text{real-of } r2)))
proof (transfer, unfold equal-real-def, clarsimp)
fix p1 q1 p2 q2 :: rat and b1 b2 :: nat
assume base: q1 = 0 ∨ b1 ∈ sqrt-irrat
assume base: q2 = 0 ∨ b2 ∈ sqrt-irrat
assume base: q1 = 0 ∨ q2 = 0 ∨ b1 = b2
let \(?l\) = \text{real-of-rat } p1 + \text{real-of-rat } q1 \ast \text{sqrt } (\text{of-nat } b1) = \text{real-of-rat } p2 + \text{real-of-rat } q2 \ast \text{sqrt } (\text{of-nat } b2)
let \(?m\) = \text{real-of-rat } q1 \ast \text{sqrt } (\text{of-nat } b1) = \text{real-of-rat } (p2 - p1) + \text{real-of-rat } q2 \ast \text{sqrt } (\text{of-nat } b2)
let \(?r\) = \text{p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)}
have \(?l\) ←→ \text{p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)}
by (auto simp: of-rat-add of-rat-diff of-rat-minus)
also have \(...\) ←→ \text{p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)}
proof
assumption
from base have q1 = 0 ∨ q1 ≠ 0 ∧ q2 = 0 ∨ q1 ≠ 0 ∧ q2 ≠ 0 ∧ b1 = b2
by auto
thus p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)
proof
assumption q1 = 0
with \(?m\) have real-of-rat (p2 - p1) + real-of-rat q2 * sqrt (of-nat b2) = 0
by auto
with sqrt-irrat b2 have q2 = 0 by auto
with q1 \(?m\) show \(?thesis\) by auto
next
assume q1 ≠ 0 ∧ q2 = 0 ∨ q1 ≠ 0 ∧ q2 ≠ 0 ∧ b1 = b2
thus \(?thesis\)
proof
assumption ass: q1 ≠ 0 ∧ q2 = 0
with ⟨?m⟩ have real-of-rat (p1 - p2) + real-of-rat q1 * sqrt (of-nat b1) = 0
  by (auto simp: of-rat-diff)
with b1 have q1 = 0 using sqrt-irrat by auto
with ass show ?thesis by auto
next
  assume ass: q1 ≠ 0 ∧ q2 ≠ 0 ∧ b1 = b2
with ⟨?m⟩ have *: real-of-rat (p2 - p1) + real-of-rat (q2 - q1) * sqrt
  (of-nat b2) = 0
  by (auto simp: field-simps of-rat-diff)
  have q2 - q1 = 0
  by (rule sqrt-irrat[of *, insert ass b2, auto])
with * ass show ?thesis by auto
qed
qed auto
finally show ℓ = ?r by simp
qed

lemma ma-floor: floor (real-of-r) = ma-floor r
proof (transfer, unfold Let-def, clarsimp)
  fix p q :: rat and b :: nat
obtain z1 n1 where qp: quotient-of p = (z1,n1) by force
obtain z2 n2 where qq: quotient-of q = (z2,n2) by force
from quotient-of-denom-pos[OF qp] have n1: 0 < n1.
from quotient-of-denom-pos[OF qq] have n2: 0 < n2.
from quotient-of-div[OF qp] have p: p = of-int z1 / of-int n1.
from quotient-of-div[OF qq] have q: q = of-int z2 / of-int n2.
  have p: p = of-int (z1 * n2) / of-int (n1 * n2) unfolding p using n2 by auto
  have q: q = of-int (z2 * n1) / of-int (n1 * n2) unfolding q using n1 by auto
  define z1n2 where z1n2 = z1 * n2
  define z2n1 where z2n1 = z2 * n1
  define n12 where n12 = n1 * n2
  define r-add where r-add = of-int (z2n1) * sqrt (real-of-int (int b))
  from n1 n2 have n120: n12 > 0 unfolding n12-def by simp
  have floor (of-rat p + of-rat q * sqrt (real-of-nat b)) = floor ((of-int z1n2 +
    r-add) / of-int n12)
    unfolding r-add-def n12-def z1n2-def
    unfolding p q add-divide-distrib of-rat-divide of-rat-of-int-eq of-int-of-nat-eq by simp
also have ... = floor (of-int z1n2 + r-add) div n12
  by (rule floor-div-pos-int[OF n120])
also have of-int z1n2 + r-add = r-add + of-int z1n2 by simp
also have floor (...) = floor r-add + z1n2 by simp
also have ... = z1n2 + floor r-add by simp
finally have id: [of-rat p + of-rat q * sqrt (of-nat b)] = (z1n2 + [r-add]) div
  n12.
  show [of-rat p + of-rat q * sqrt (of-nat b)] =
    (case quotient-of p of
(\(z_1, n_1\)) ⇒

\[\text{case quotient-of q of (z_2, n_2) ⇒ (z_1 \cdot n_2 + (\text{if } 0 \leq z_2 \cdot n_1 \text{ then } \text{sqrt-int-floor-pos} (z_2 \cdot n_1 \cdot (z_2 \cdot n_1) \cdot \text{int b}) \text{ else } - \text{sqrt-int-ceiling-pos} (z_2 \cdot n_1 \cdot (z_2 \cdot n_1) \cdot \text{int b}))) \div (n_1 \cdot n_2))\]

unfolding qp qq split id n12-def z1n2-def

proof

have \(\text{ge-int} : z_2 \cdot n_1 \cdot (z_2 \cdot n_1) \cdot \text{int b} \geq 0\)

by (metis mult-nonneg-nonneg zero-le-square of-nat-0-le-iff)

show \(\lfloor r \cdot \text{add} \rfloor = (\text{if } 0 \leq z_2 \cdot n_1 \text{ then } \text{sqrt-int-floor-pos} (z_2 \cdot n_1 \cdot (z_2 \cdot n_1) \cdot \text{int b}) \text{ else } - \text{sqrt-int-ceiling-pos} (z_2 \cdot n_1 \cdot (z_2 \cdot n_1) \cdot \text{int b})))\)

proof (cases \(z_2 \cdot n_1 \geq 0\))

case True

hence \(\text{ge} : \text{real-of-int} (z_2 \cdot n_1) \geq 0\)

by (metis of-int-0-le-iff)

have \(\text{r-add} = \text{sqrt} (\text{of-int} (z_2 \cdot n_1 \cdot (z_2 \cdot n_1) \cdot \text{int b}))\)

unfolding \(\text{r-add-def} z2n1-def\) using \(\text{sqrt}-\text{sqrt}\) [OF \(\text{ge}\)]

by (simp add: ac-simps real-sqrt-mult)

show \(?\text{thesis}\) unfolding \(\text{radd-sqrt-int-floor-pos}\) [OF \(\text{ge-int}\)] using True by simp

next

case False

hence \(\text{ge} : \text{real-of-int} (- (z_2 \cdot n_1)) \geq 0\)

by (metis mult-zero-left neg-0-le-iff-le of-int-0-le-iff order-refl zero-le-mult-iff)

have \(\text{r-add} = - \text{sqrt} (\text{of-int} (z_2 \cdot n_1 \cdot (z_2 \cdot n_1) \cdot \text{int b}))\)

unfolding \(\text{r-add-def} z2n1-def\) using \(\text{sqrt}-\text{sqrt}\) [OF \(\text{ge}\)]

by (metis minus-minus minus-mult-commute minus-mult-right of-int-minus of-int-mult real-sqrt-minus real-sqrt-mult z2n1-def)

hence \(\text{radd} : \text{floor} \text{ r-add} = - \text{ceiling} (\text{sqrt} (\text{of-int} (z_2 \cdot n_1 \cdot (z_2 \cdot n_1) \cdot \text{int b})))\)

by (metis ceiling-def minus-minus)

show \(?\text{thesis}\) unfolding \(\text{radd-sqrt-int-ceiling-pos}\) [OF \(\text{ge-int}\)] using False by simp

qed

qed

lemma comparison-impl:

\((x \cdot \text{real}) \leq (y \cdot \text{real}) = \text{ge-0} \; (y - x)\)

\((x \cdot \text{real}) < (y \cdot \text{real}) = (x \neq y \land \text{ge-0} \; (y - x))\)

by (simp-all add: \(\text{ge-0-def}\), linarith+)

lemma ma-of-rat: \(\text{real-of-rat} \; r = \text{real-of} \; (\text{ma-of-rat} \; r)\)

by (transfer, auto)

definition is-rat :: \(\text{real} \Rightarrow \text{bool}\) where

[\text{code-abbrev}]: \(\text{is-rat} \; x \iff x \in \mathbb{Q}\)
lemma ma-is-rat: is-rat (real-of x) = ma-is-rat x
proof (transfer, unfold is-rat-def, clarsimp)
  fix p q :: rat and b :: nat
  let ?p = real-of-rat p
  let ?q = real-of-rat q
  let ?b = real-of-rat b
  let ?b' = real-of-rat (of-nat b)
  assume b: q = 0 ∨ b ∈ sqrt-irrat
  show (?p + ?q * sqrt ?b ∈ Q) = (q = 0)
  proof (cases q = 0)
    case False
    from False b have b: b ∈ sqrt-irrat by auto
    { assume ?p + ?q * sqrt ?b ∈ Q
      from this[unfolded Rats-def] obtain r where r: ?p + ?q * sqrt ?b = real-of-rat r
      by auto
      let ?r = real-of-rat r
      from r have real-of-rat (p − r) + ?q * sqrt ?b = 0 by (simp add: of-rat-diff)
      from sqrt-irrat[OF disjI2[OF b] this] False have False by auto
    }
    thus ?thesis by auto
  qed auto
qed auto

definition sqrt-real x = (if x ∈ Q ∧ x ≥ 0 then (if x = 0 then [0] else (let sx = sqrt x in [sx,−sx])) else [])

lemma sqrt-real[simp]: assumes x: x ∈ Q
  shows set (sqrt-real x) = {y. y * y = x}
proof (cases x ≥ 0)
  case False
  hence ∃ y. y * y ≠ x by auto
  with False show ?thesis unfolding sqrt-real-def by auto
next
  case True
  thus ?thesis using x
  by (cases x = 0, auto simp: Let-def sqrt-real-def)
qed

code_datatype real-of

declare [[code drop]:
  plus :: real ⇒ real ⇒ real
  uminus :: real ⇒ real
  times :: real ⇒ real ⇒ real
  inverse :: real ⇒ real
  floor :: real ⇒ int
  sqrt
\[ \text{HOL.equal} :: \text{real} \Rightarrow \text{real} \Rightarrow \text{bool} \]

\textbf{lemma [code]}:
\begin{quote}
\begin{center}
\begin{tabular}{l}
\text{Ratreal} = \text{real-of} \circ \text{ma-of-rat} \\
\text{by} \ (\text{simp add: fun-eq-iff}) (\text{transfer, simp})
\end{tabular}
\end{center}
\end{quote}

\textbf{lemmas ma-code-eqns [code equation]} = \text{ma-ge-0 ma-floor ma-0 ma-1 ma-uminus ma-inverse ma-sqrt ma-plus ma-times ma-equal ma-is-rat comparison-impl}

\textbf{lemma [code equation]}:
\begin{quote}
\begin{center}
\begin{tabular}{l}
\((x :: \text{real}) / (y :: \text{real}) = x \ast \text{inverse} y \\
\((x :: \text{real}) - (y :: \text{real}) = x + (- y) \\
\text{by} \ (\text{simp-all add: divide-inverse})
\end{tabular}
\end{center}
\end{quote}

Some tests with small numbers. To work on larger number, one should additionally import the theories for efficient calculation on numbers

\begin{quote}
\begin{center}
\begin{tabular}{l}
\text{value} \ [101.1 \ast (3 \ast \text{sqrt} 2 + 6 \ast \text{sqrt} 0.5)] \\
\text{value} \ [606.2 \ast \text{sqrt} 2 + 0.001] \\
\text{value} \ 101.1 \ast (3 \ast \text{sqrt} 2 + 6 \ast \text{sqrt} 0.5) = 606.2 \ast \text{sqrt} 2 + 0.001 \\
\text{value} \ 101.1 \ast (3 \ast \text{sqrt} 2 + 6 \ast \text{sqrt} 0.5) > 606.2 \ast \text{sqrt} 2 + 0.001 \\
\text{value} \ (\text{sqrt} 0.1 \in \mathbb{Q}, \text{sqrt} (- 0.09) \in \mathbb{Q})
\end{tabular}
\end{center}
\end{quote}

end

\section{A unique representation of real numbers via triples}

\textbf{theory} \text{Real-Unique-Impl}

\textbf{imports}
\begin{quote}
\begin{center}
\begin{tabular}{l}
\text{Prime-Product} \\
\text{Real-Impl} \\
\text{Show.Show-Instances} \\
\text{Show.Show-Real}
\end{tabular}
\end{center}
\end{quote}

\textbf{begin}

We implement the real numbers again using triples, but now we require an additional invariant on the triples, namely that the base has to be a prime product. This has the consequence that the mapping of triples into \(\mathbb{R}\) is injective. Hence, equality on reals is now equality on triples, which can even be executed in case of different bases. Similarly, we now also allow different basis in comparisons. Ultimately, injectivity allows us to define a show-function for real numbers, which pretty prints real numbers into strings.

\textbf{typedef} \text{mini-alg-unique} = 
\begin{quote}
\begin{center}
\begin{tabular}{l}
\{ \ r :: \text{mini-alg} . \ \text{ma-coeff} \ r = 0 \land \text{ma-base} \ r = 0 \lor \text{ma-coeff} \ r \neq 0 \land \text{prime-product} (\text{ma-base} \ r) \}\n\end{tabular}
\end{center}
\end{quote}

\text{by} \ (\text{transfer, auto})
setup-lifting type-definition-mini-alg-unique

lift-definition real-of-u :: mini-alg-unique ⇒ real is real-of .
lift-definition mau-floor :: mini-alg-unique ⇒ int is mau-floor .
lift-definition mau-of-rat :: rat ⇒ mini-alg-unique is mau-of-rat by (transfer, auto)
lift-definition mau-rat :: mini-alg-unique ⇒ rat is mau-rat .
lift-definition mau-base :: mini-alg-unique ⇒ nat is mau-base .
lift-definition mau-coeff :: mini-alg-unique ⇒ mini-alg-unique is mau-coeff by (transfer, auto)
lift-definition mau-uminus :: mini-alg-unique ⇒ mini-alg-unique is mau-uminus
by (transfer, auto)
lift-definition mau-compatible :: mini-alg-unique ⇒ mini-alg-unique ⇒ bool is mau-compatible .
lift-definition mau-ge-0 :: mini-alg-unique ⇒ bool is mau-ge-0 .
lift-definition mau-inverse :: mini-alg-unique ⇒ mini-alg-unique is mau-inverse
by (transfer, auto simp: ma-normalize-def Let-def split: if-splits)
lift-definition mau-plus :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique
by (transfer, auto simp: ma-normalize-def split: if-splits)
lift-definition mau-times :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique
by (transfer, auto simp: ma-normalize-def split: if-splits)
lift-definition ma-identity :: mini-alg ⇒ mini-alg ⇒ bool is (=) .
lift-definition mau-equal :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique ⇒ bool is mau-equal .
lift-definition mau-is-rat :: mini-alg-unique ⇒ bool is mau-is-rat .

lemma Ratreal-code[code]:
  Ratreal = real-of-u ◦ mau-of-rat
by (simp add: fun-eq-iff) (transfer, transfer, simp)

lemma mau-floor: floor (real-of-u r) = mau-floor r
using mau-floor by (transfer, auto)
lemma mau-inverse: inverse (real-of-u r) = real-of-u (mau-inverse r)
using mau-inverse by (transfer, auto)
lemma mau-uminus: − (real-of-u r) = real-of-u (mau-uminus r)
using mau-uminus by (transfer, auto)
lemma mau-times:
  (real-of-u r1 * real-of-u r2) = (if mau-compatible r1 r2
  then real-of-u (mau-times r1 r2) else
  Code.abort (STR "different base") (λ -. real-of-u r1 * real-of-u r2))
using mau-times by (transfer, auto)
lemma mau-plus:
  (real-of-u r1 + real-of-u r2) = (if mau-compatible r1 r2
  then real-of-u (mau-plus r1 r2) else
  Code.abort (STR "different base") (λ -. real-of-u r1 + real-of-u r2))
using mau-plus by (transfer, auto)
lemma real-of-u-inj[simp]: real-of-u x = real-of-u y \iff x = y

proof
  note field-simps[simp] of-rat-diff[simp]
  assume real-of-u x = real-of-u y
  thus x = y

  proof (transfer)
    fix x y
    assume ma-coeff x = 0 \and ma-base x = 0 \or ma-coeff x \neq 0 \and prime-product (ma-base x)
    and ma-coeff y = 0 \and ma-base y = 0 \or ma-coeff y \neq 0 \and prime-product (ma-base y)
    and real-of x = real-of y
    thus x = y

    proof (transfer, clarsimp)
      fix p1 q1 p2 q2 :: rat and b1 b2
      let ?p1 = real-of-rat p1
      let ?p2 = real-of-rat p2
      let ?q1 = real-of-rat q1
      let ?q2 = real-of-rat q2
      let ?b1 = real-of-nat b1
      let ?b2 = real-of-nat b2
      assume q1: q1 = 0 \and b1 = 0 \or q1 \neq 0 \and prime-product b1
      and q2: q2 = 0 \and b2 = 0 \or q2 \neq 0 \and prime-product b2
      and i1: q1 = 0 \or b1 \in sqrt-irrat
      and i2: q2 = 0 \or b2 \in sqrt-irrat
      show p1 = p2 \and q1 = q2 \and b1 = b2

      proof (cases q1 = 0)
        case True
        have q2 = 0
          by (rule sqrt-irrat[OF i2, of p2 − p1], insert eq True q1, auto)
        with True q1 q2 eq show ?thesis by auto

      next
      case False
      hence 1: q1 \neq 0 prime-product b1 using q1 by auto

      { assume *: q2 = 0
        have q1 = 0
          by (rule sqrt-irrat[OF i1, of p1 − p2], insert eq * q2, auto)
        with False have False by auto
      }
      hence 2: q2 \neq 0 prime-product b2 using q2 by auto

      from 1 i1 have b1: b1 \neq 0 unfolding sqrt-irrat-def by (cases b1, auto)
      from 2 i2 have b2: b2 \neq 0 unfolding sqrt-irrat-def by (cases b2, auto)
      let ?sq = λ x. x * x
      define q3 where q3 = p2 − p1
      let ?q3 = real-of-rat q3
      let ?c = of-rat (q2 * q2 * of-nat b2 + ?sq q3 − ?sq q1 * of-nat b1) + of-rat (2 * q2 * q3) * sqrt ?b2

      21
from eq have *: ?q1 * sqrt ?b1 = ?q2 * sqrt ?b2 + ?q3 
  by (simp add: q3-def)
from arg-cong[OF this, of ?sq] have 0 = (real-of-rat 2 * ?q2 * ?q3) * sqrt ?b2 + 
  by auto
also have ... = ?e 
  by (simp add: of-rat-mult of-rat-add of-rat-minus)
finally have eq: ?e = 0 by simp
from sqrt-irrat[OF this] 2 i2 have q3: q3 = 0 by auto
hence p: p1 = p2 unfolding q3-def by simp
let ?b1 = rat-of-nat b1
let ?b2 = rat-of-nat b2
from eq[unfolded q3] have eq: ?sq ?q2 * ?b2 = ?sq q1 * ?b1 by auto
obtain z1 n1 where d1: quotient-of q1 = (z1,n1) by force
obtain z2 n2 where d2: quotient-of q2 = (z2,n2) by force
note id = quotient-of-div[OF d1] quotient-of-div[OF d2]
note pos = quotient-of-denom-pos[OF d1] quotient-of-denom-pos[OF d2]
from id(1) 1(1) pos(1) have z1: z1 ≠ 0 by auto
from id(2) 2(1) pos(2) have z2: z2 ≠ 0 by auto
let ?n1 = rat-of-int n1
let ?n2 = rat-of-int n2
let ?z1 = rat-of-int z1
let ?z2 = rat-of-int z2
from arg-cong[OF eq[simplified id], of λ x. x * ?sq ?n1 * ?sq ?n2, 
  simplified field-simps]
  have ?sq (?n1 * ?z2) * ?b2 = ?sq (?n2 * ?z1) * ?b1 
  using pos by auto
moreover have ?n1 * ?z2 ≠ 0 ?n2 * ?z1 ≠ 0 using z1 z2 pos by auto
ultimately obtain i1 i2 where 0: rat-of-int i1 ≠ 0 rat-of-int i2 ≠ 0 
  and eq: ?sq (rat-of-int i2) * ?b2 = ?sq (rat-of-int i1) * ?b1 
  unfolding of-int-mult[symmetric] by blast+
let ?b1 = int b1
let ?b2 = int b2
from eq have eq: ?sq i1 * ?b1 = ?sq i2 * ?b2 
  by (metis (hide-lams, no-types) of-int-eq-iff of-int-mult of-int-of-nat-eq)
from 0 have 0: i1 ≠ 0 i2 ≠ 0 by auto
from arg-cong[OF eq, of nat] have ?sq (nat (abs i1)) * b1 = ?sq (nat (abs 
  i2)) * b2 
  by (metis abs-of-nat eq nat-abs-mult-distrib nat-int)
moreover have nat (abs i1) > 0 nat (abs i2) > 0 using 0 by auto
ultimately obtain n1 n2 where 0: n1 > 0 n2 > 0 and eq: ?sq n1 * b1 
  = ?sq n2 * b2 by blast
from b1 0 have b1: b1 > 0 n1 > 0 n1 * n1 > 0 by auto
from b2 0 have b2: b2 > 0 n2 > 0 n2 * n2 > 0 by auto
{ 
  fix p :: nat assume p: prime p 
  have multiplicity p (?sq n1 * b1) = multiplicity p b1 + 2 * multiplicity p 
  n1 

22
using \( b_1 \) \( p \) \( \text{by} \) (auto \( \text{simp: prime-elem-multiplicity-mult-distrib} \))
also have \( \ldots \mod 2 = \text{multiplicity} \ p \ b_1 \mod 2 \) \( \text{by} \) presburger
finally have \( \text{id1: multiplicity} \ p \ (?sq \ n1 \ast b1) \mod 2 = \text{multiplicity} \ p \ b1 \mod 2 \).
have \( \text{multiplicity} \ p \ (?sq \ n2 \ast b2) = \text{multiplicity} \ p \ b2 + 2 \ast \text{multiplicity} \ p \ n2 \)
using \( b_2 \) \( p \) \( \text{by} \) (auto \( \text{simp: prime-elem-multiplicity-mult-distrib} \))
also have \( \ldots \mod 2 = \text{multiplicity} \ p \ b_2 \mod 2 \) \( \text{by} \) presburger
finally have \( \text{id2: multiplicity} \ p \ (?sq \ n2 \ast b2) \mod 2 = \text{multiplicity} \ p \ b2 \mod 2 \).
from \( \text{id1 id2 eq have eq: multiplicity} \ p \ b1 \mod 2 = \text{multiplicity} \ p \ b2 \mod 2 \) \( \text{by} \) simp
from \( 1(2) \ 2(2) \ p \) \( \text{have multiplicity} \ p \ b1 \leq 1 \) \( \text{multiplicity} \ p \ b2 \leq 1 \)
unfolding \( \text{prime-product-def} \) \( \text{by} \) auto
with \( \text{eq have multiplicity} \ p \ b1 = \text{multiplicity} \ p \ b2 \) \( \text{by} \) simp
}\}
with \( b1(1) \ b2(1) \) \( \text{have b: b1 = b2 by} \) (rule \( \text{multiplicity-eq-nat} \))
from \( \ast[\text{unfolded b q3}] b1(1) \ b2(1) \) \( \text{have q: q1 = q2 by} \) simp
from \( p q b \) \( \text{show} \) \( \text{thesis by} \) blast
qed
qed
qed
qed
simp

lift-definition \( \text{mau-sqrt :: mini-alg-unique \Rightarrow mini-alg-unique is} \)
\( \lambda \ \text{ma. let} \ (a,b) = \text{quotient-of} \ (\text{ma-rat ma}); \ (\text{sq, fact}) = \text{prime-product-factor} \)
(\( \text{n} \text{at} \ (\text{abs a} \ast b) \))
\( \text{ma'} = \text{ma-of-rat} \ (\text{of-int} \ (\text{sgn(a)}) \ast \text{of-nat sq} \ / \text{of-int b}) \)
in \( \text{ma-times ma'} \ (\text{ma-sqrt (ma-of-rat (of-nat fact))}) \)
proof --
fix \( \text{ma :: mini-alg} \)
let \( ?\text{num} = \)
let \( (a, b) = \text{quotient-of} \ (\text{ma-rat ma}); \ (\text{sq, fact}) = \text{prime-product-factor} \)
(\( \text{n} \text{at} \ (|a| \ast b) \))
\( \text{ma'} = \text{ma-of-rat} \ (\text{rat-of-int} \ (\text{sgn a}) \ast \text{rat-of-nat sq} \ / \text{of-int b}) \)
in \( \text{ma-times ma'} \ (\text{ma-sqrt (ma-of-rat (of-nat fact))}) \)
obtain \( a \ b \) where \( q: \text{quotient-of} \ (\text{ma-rat ma}) = (a,b) \) \( \text{by} \) force
obtain \( \text{sq fact where} \ q \text{pf: prime-product-factor} \ (\text{nat} \ (\text{abs a} \ast b)) = (\text{sq, fact}) \)
by force
define \( \text{asq where} \ \text{asq = rat-of-int} \ (\text{sgn a}) \ast \text{of-nat sq} \ / \text{of-int b} \)
define \( \text{ma' where ma'} = \text{ma-of-rat asq} \)
define \( \text{sqrt where} \ \text{sqrt = ma-sqrt (ma-of-rat (rat-of-nat fact))} \)
have \( \text{num: ?num = ma-times ma'sqrt unfolding q ppf asq-def Let-def split ma'-def sqrt-def} \) ..
let \( ?\text{inv} = \text{\lambda ma. ma-coeff ma} = 0 \land \text{ma-base ma} = 0 \lor \text{ma-coeff ma} \neq 0 \land \)
\( \text{prime-product (ma-base ma)} \)
have \( \text{ma': ?inv ma' unfolding ma'-def} \)
by (transfer, auto)
have \( \text{id: i \cdot int i * 1 = i \land i :: rat. i / 1 = i \text{ rat-of-int 1 = 1 inverse} \ (1 ::} \)
rat) = 1

\[ n \cdot \text{nat} \mid \text{int} n \mid n \text{ by auto} \]

from prime-product-factor[OF ppf] have prime-product fact by auto

hence sqrt': ?inv sqrt unfolding sqrt-def

by (transfer, unfold split quotient-of-int Let-def id, case-tac sqrt-int | int facta],

auto)

show ?inv ?num unfolding num using ma' sqrt

by (transfer, auto simp: ma-normalize-def split: if-splits)

qed

lemma sqrt-sgn[simp]: sqrt (of-int (sgn a)) = of-int (sgn a)

by (cases a ≥ 0, cases a = 0, auto simp: real-sqrt-minus)

lemma mau-sqrt-main: mau-coeff r = 0 \implies sqrt (real-of-u r) = real-of-u (mau-sqrt r)

proof (transfer)

fix r

assume ma-coeff r = 0

hence rr: real-of r = of-rat (ma-rat r) by (transfer, auto)

obtain a b where q: quotient-of (ma-rat r) = (a,b) by force

from quotient-of-div[OF q] have r: ma-rat r = of-rat a / of-rat b by auto

from quotient-of-denom-pos[OF q] have b: b > 0 by auto

obtain sq fact where ppf: prime-product-factor (nat (|a| * b)) = (sq, fact) by force

from prime-product-factor[OF ppf] have ab: nat (|a| * b) = sq * sq * fact ..

have sqrt (real-of r) = sqrt(of-int a / of-int b) unfolding rr r

by (metis of-rat-divide of-rat-of-int-eq)

also have real-of-int a / of-int b = of-rat a / of-rat b by (of-int b * of-int b)

using b by auto

also have sqrt (...) = sqrt (of-int a * of-int b) / of-int b using sqrt-sqrt[of real-of-int b] b

by (metis less-eq-real-def of-int-0-less-iff real-sqrt-divide real-sqrt-mult)

also have real-of-int a * of-int b = real-of-int (a * b) by auto

also have a * b = sgn a * (abs a * b) by (simp, metis mult-sgn-abs)

also have real-of-int (...) = of-int (sgn a) * real-of-int (|a| ^ 2)

unfolding of-int-mult[of sgn a] ..

also have real-of-int (|a| * b) = of-nat (nat (abs a * b)) unfolding b

by (metis abs-sgn mult-pos-pos mult-zero-left nat-int_of-nat-eq of-nat-0

zero-less-iff-zero-less-imp-eq-int)

also have ... = of-nat fact * (of-nat sq * of-nat sq) unfolding ab of-nat-mult

by simp

also have sqrt (of-int (sgn a) * (of-int fact * (of-nat sq * of-nat sq))) =

of-int (sgn a) * sqrt (of-int fact) * of-int sq

unfolding real-sqrt-mult by simp

finally have r: sqrt (real-of r) = real-of-int (sgn a) * real-of-int sq / real-of-int b

by (simp)

let ?asqb = ma-of-rat (rat-of-int (sgn a) * rat-of-int sq / rat-of-int b)

let ?f = ma-of-rat (rat-of-int fact)

let ?sq = ma-sqrt ?f
have sq: 0 ≤ ma-rat ?f ma-coeff ?f = 0 by ((transfer, simp)+)
have compat: ∨ m. (ma-compatible ?asqb m) = True
  by (transfer, auto)
show sqrt (real-of r) =
  real-of
  (let (a, b) = quotient-of (ma-rat r); (sq, fact) = prime-product-factor (nat
  ([a] * b));
  ma' = ma-of-rat (rat-of-int (sgn a) * rat-of-int sq / rat-of-int b)
  in ma-times ma' (ma-sqrt (ma-of-rat (rat-of-int fact))))
unfolding q ppf Let-def split
unfolding r
unfolding ma-times[ symmetric, of ?asqb, unfolded compat if-True]
unfolding ma-sqrt-main [OF sq, symmetric]
unfolding ma-of-rat[ symmetric]
by (simp add: of-rat-divide of-rat-mult)
qed

lemma mau-sqrt: sqrt (real-of-u r) = (if mau-coeff r = 0 then
  real-of-u (mau-sqrt r)
else Code.abort (STR "cannot represent sqrt of irrational number") (λ _. sqrt
  (real-of-u r)))
  using mau-sqrt-main[of r] by (cases mau-coeff r = 0, auto)

lemma mau-0: 0 = real-of-u (mau-of-rat 0)
  using mau-0 by (transfer, auto)

lemma mau-1: 1 = real-of-u (mau-of-rat 1)
  using mau-1 by (transfer, auto)

lemma mau-equal:
  HOL.equal (real-of-u r1) (real-of-u r2) = mau-equal r1 r2
  unfolding equal-real-def
  using real-of-u-inj[of r1 r2]
  by (transfer, transfer, auto)

lemma mau-ge-0: ge-0 (real-of-u x) = mau-ge-0 x
  using mau-ge-0
  by (transfer, auto)

definition real-lt :: real ⇒ real ⇒ bool where real-lt = (<)

The following code equation terminates if it is started on two different
inputs.

lemma real-lt [code equation]: real-lt x y = (let fx = floor x; fy = floor y in
  (if fx < fy then True else if fx > fy then False else real-lt (x * 1024) (y * 1024)))
proof (cases floor x < floor y)
  case True
  thus ?thesis by (auto simp: real-lt-def floor-less-cancel)
next
  case False
  note nless = this
  show ?thesis
  proof (cases floor x > floor y)
    case True
For comparisons we first check for equality. Then, if the bases are compatible we can just compare the differences with 0. Otherwise, we start the recursive algorithm real-lt which works on arbitrary bases. In this way, we have an implementation of comparisons which can compare all representable numbers.

Note that in Real-Impl.Real-Impl we did not use real-lt as there the code-equations for equality already require identical bases.

**lemma** comparison-impl:
\[
\text{real-of-u } x \leq \text{real-of-u } y \iff \text{real-of-u } x = \text{real-of-u } y \lor \\
\text{real-of-u } x < \text{real-of-u } y \iff \text{real-of-u } x \neq \text{real-of-u } y \land
\]
\[
\text{real-of-u } x < \text{real-of-u } y \iff \text{real-of-u } x \neq \text{real-of-u } y \land
\]

**unfolding** ge-0-def real-lt-def by (auto simp del: real-of-u-inj)

**lemma** mau-is-rat: is-rat (real-of-u x) = mau-is-rat x using mau-is-rat
by (transfer, auto)

**lift-definition** mau-show-real :: mini-alg-unique ⇒ string is
\[
\lambda (p,q,b). \text{let } sb = \text{shows "sqrt" o shows b o shows "" }; \\
\text{qb } = (\text{if q } = 1 \text{ then } sb \text{ else if q } = -1 \text{ then shows "-" o sb \text{ else shows q o shows "**" o sb} }) \text{ in} \\
\text{if q } = 0 \text{ then shows p [] else} \\
\text{if p } = 0 \text{ then qb [] else} \\
\text{if q } < 0 \text{ then ((shows p o qb) []) else ((shows p o shows "**" o qb) [])} .
\]

**lift-definition** mau-show-real :: mini-alg-unique ⇒ string is mau-show-real .

**overloading** show-real ≡ show-real
begin
**definition** show-real
\[
\text{where } \text{show-real } x \equiv \\
\text{(if } \exists \ y . x = \text{real-of-u } y \text{ then mau-show-real (THE y. x = real-of-u y) else}
\]
end

**lemma** mau-show-real: show-real (real-of-u x) = mau-show-real x
**unfolding** show-real-def by simp
code-datatype real-of-u

declare [
  code drop:
  plus :: real ⇒ real ⇒ real
  uminus :: real ⇒ real
  times :: real ⇒ real ⇒ real
  inverse :: real ⇒ real
  floor :: real ⇒ int
  sqrt
  HOL.equal :: real ⇒ real ⇒ bool
  ge-0
  is-rat
  less :: real ⇒ real ⇒ bool
  less-eq :: real ⇒ real ⇒ bool
]

lemmas mau-code-eqns [code] = mau-floor mau-0 mau-1 mau-uminus mau-inverse
  mau-sqrt mau-plus mau-times mau-equal mau-ge-0 mau-is-rat
  mau-show-real comparison-impl

  Some tests with small numbers. To work on larger number, one should additionally import the theories for efficient calculation on numbers

value [101.1 * (sqrt 18 + 6 * sqrt 0.5)]
value [324 * sqrt 7 + 0.001]
value 101.1 * (sqrt 18 + 6 * sqrt 0.5) = 324 * sqrt 7 + 0.001
value 101.1 * (sqrt 18 + 6 * sqrt 0.5) > 324 * sqrt 7 + 0.001
value show (101.1 * (sqrt 18 + 6 * sqrt 0.5))
value (sqrt 0.1 ∈ Q, sqrt (− 0.09) ∈ Q)

end

Acknowledgements

We thank Bertram Felgenhauer for interesting discussions and especially for mentioning Cauchy’s mean theorem during the formalization of the algorithms for computing roots.

References


